Exercises Week #3: Programming I

Notes:
Please solve the following problems by implementing short Python functions based on the template `template03.py` available on the course webpage or the moodle page. Copy the template to your Python folder and rename it such that the new file name consists of your last name and the exercise number (e.g. `yourname03.py`). Open the template in an IDE such as Spyder. Enter your name and your student ID in the first comment lines. Follow the instructions given in the comments and enter your code where indicated. Test your programs by running the entire script. Submit the Python file with your solutions by e-mail to `image.processing.jena@gmail.com`

Solutions have to be submitted before November 24th, 2020 8.00 a.m. (CET)

Exercise 1 – "Hello, World!" and "99 bottles of beer" (Level: easy)

Task 1A. The purpose of a "Hello, World!" program is to illustrate the basic syntax of a programming language.[1] Please implement a program that generates the message "Hello, World!". Proceed as follows:

1. The template contains a function `hello_world`. When you run your script and enter `1A` followed by Enter, this function will be called. That way, you can check if your program is correct. The same holds for the other exercises and tasks.

2. The built-in function `print` can be used to generate output. For example, `print(a)` displays the value of a variable `a` (without displaying its name). `print` can be also used to output a string that can be passed as a variable or as text enclosed in quotation marks. In this way messages can be displayed to the user. For example, `print('Hello !')` generates the message

   ```
   Hello !
   ```

   To display a message and a value in the same line, the value must be first converted to a string by using the function `int(A)`. For instance, the lines:

   ```
   a = 3
   print('The value of variable ''a'' is ' + str(a))
   ```

   generate the output: `The value of variable ''a'' is 3`

   A more elegant version is to use *format strings* (more on this in the next lecture):

[1] See helloworldcollection.github.io for a large collection of examples.

```
print('The value of variable ''a'' is {0}'.format(a))
```

3. Save the modified template in the current directory.
4. Run the script and enter `1A` upon the prompt requesting you to pick an exercise. The message

   ```
   Hello, World!
   ```

   should now appear on the command line.

Task 1B. Many computer programs produce the lyrics of the song "99 bottles of beer". In contrast to the "Hello, World!" program, different looping constructs can be compared in this way.[2] The song goes as follows:

99 bottles of beer on the wall, 99 bottles of beer.
Take one down, pass it around,

98 bottles of beer on the wall, 98 bottles of beer.
Take one down, pass it around,

⋮

1 bottle of beer on the wall, 1 bottle of beer.
Take it down, pass it around,

No more bottles of beer on the wall, no more bottles of beer.
Go to the store and buy some more.

Write a program that generates this output.

Exercise 2 – Adding up numbers (Level: medium)

A well-known story about *Carl-Friedrich Gauß* goes as follows: While still in primary school, the teacher asked his class to add up all numbers from 1 to 100, assuming that this would keep the class busy for quite a while. He was stunned when young Carl-Friedrich presented the answer, only after a few seconds of thought: **5050**. The teacher couldn't understand how Gauß had calculated the sum so quickly in his head. But the eight years old explained that the problem was in fact quite simple...

You will add up a series of numbers with the help of Python. The functions that you will implemented are called `addup(start, end)` where the first variable `start` specifies the first number in the series and the second variable `end` is the last number.

[2] See https://en.wikipedia.org/wiki/99_Bottles_of_Beer for more information.

Task 2A. The Python built-in function `sum(a)` calculates the sum of a list `a`. The built-in function `range(1, 10)` allows you to generate a list `[1, 2, ..., 9]`. Write a function `addup(start, end)` that uses the Python functions `sum` and `range` to solve the problem.

Task 2B. Write a function `addup2(start, end)` that adds up all numbers from `start` to `end` by using a `for` loop.

Task 2C. Write a function `addup3(start, end)` that adds up all numbers from `start` to `end` by using a `while` loop.

Task 2D. Gauß arrived at his answer so quickly by pairing the numbers from each end and working from the ends towards the middle. In other words,

$$1 + 2 + 3 + ... + 98 + 99 + 100$$

can be written as

$$(1 + 100) + (2 + 99) + (3 + 98) + ... + (50 + 51)$$

Each of the 50 pairs sums up to 101. Thus the overall sum is $50 \times 101 = 5050$. Write a function `addup4(start, end)` that calculates the sum of a series directly. Test your program for an even and odd number of elements.

Exercise 3 – Conversion between decimal and binary numbers (Level: medium)

This exercise develops two functions to convert an unsigned decimal number to an unsigned binary number and back by using the successive *halving* and *doubling* methods. Use a variable of type `int` to represent the decimal number. Since no built-in type for binary numbers exists in Python, we use *bit strings* to represent binary numbers. A bit string is composed of 0's and 1's and of type `str`. That is, the statement `x = '0101'` creates a bit string representing the decimal number 5 and assigns it to a variable named `x`.

Task 3A. Implement the successive halving method (see slide 41 in lecture 2) as a Python function `halving(decimal)`. This function takes a non-negative integer as input and returns a bit string representing the binary number version of the input obtained with successive halving.

Task 3B. Implement the successive doubling method (see slide 42 in lecture 2) as a Python function `doubling(binary)` where the input is a bit string. This function takes a bit string and converts it to an integer representing the decimal number by using the successive doubling method. So if your functions work correctly, the statement `doubling(halving(1234))` should give you back `1234`.